Remarks

Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 21-26 and 29-31 are pending in the application. Claims 21-26 and 29-31 are rejected. No claims have been allowed. Claims 21, 25, and 29 are independent. Claims 21, 25, 29, and 31 have been amended.

Cited Art

The Action cites:

Srivastava et al., "Effectively Prioritizing Tests in Development Environment," February 2002, MSR-TR-2002-15, Publisher: *Association for Computing Machinery, Inc.* ("Srivastava_2"); and

Srivastava et al., "Vulcan: Binary transformation in a distributed environment," April 2001, Technical Report MSR-TR-2001-50 (Srivastava).

Claim Objections

The Action objects to various language in independent claims 1 and 27. Claims 1 and 27 are hereby cancelled without prejudice, rendering the objections moot.

Moot Claim Rejections

The Action rejects Claims 1-20, 27, and 28 under 35 USC § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. The Action also rejects claims 1-20, 27, 28, and 32-36 under 35 U.S.C § 103(a) as unpatentable over Srivastava in view of Srivastava_2. Claims 1-20, 27, 28, and 32-36 are hereby cancelled without prejudice, rendering these rejections moot.

Claim Rejections under 35 USC § 102

The Action rejects claims 21-26 and 29-31 under 35 USC 102(b) as being anticipated by Srivastava_2. Applicants respectfully submit the claims are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (*See* MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn. Claims 21, 25, and 29 are independent.

Claim 21

Claim 21, as amended recites:

> *propagating dependency information about the set of binary files* to determine subsystem dependency information for one or more subsystems which contain binary files out of the set of binary files by:
>
> *generating pairs of entry and exit points for binary files which are dependent on one another;* and
>
> determining subsystem dependency information comprising entry and exit points for the one or more subsystems based on the generated pairs of entry and exit points for the set of binary files;
>
> . . .
>
> *marking changed logical abstractions in one or more changed binary files;*
>
> *marking unchanged logical abstractions which are in one or more unchanged binary files, the unchanged logical abstractions being dependent on marked changed logical abstractions in the one or more changed binary files;*
>
> comparing test coverage to marked changed logical abstractions and to marked unchanged logical abstractions; and
>
> prioritizing tests based on maximum test coverage of marked changed logical abstractions and marked unchanged logical abstractions;
>
> performing the prioritized tests according to test priorities to produce test results.

[Emphasis added.] The claim language has been amended to clarify operation which utilizes dependencies over multiple files. For example, the Abstract notes how dependencies are determined *across* multiple files:

> A method or system collects and propagates information about dependency between logical abstractions within a binary file (e.g., basic block, procedure, etc.), dependency between binary files, and dependency between subsystems (e.g., programs, component libraries, system services, etc,)

[Application, at Abstract; emphasis added.] The Application, in Example 1, explains some motivation for this:

> *A change 110 may have very localized effects on its subsystem 104, for example, when other binary files in the subsystem 104 call the binary file containing the change 110. In other cases, a change 110 affects one or more other subsystems 102, 106, 108, for example, when a binary file 118 in the dependent subsystem 108 calls on a binary file 110 containing change. A subsystem may depend directly or indirectly on a binary file containing the change.* A binary file 118 may depend directly on a binary file 110 in another subsystem if it calls 116 the binary file. Other dependence is not so apparent. For example, a binary file 118 may call a binary file 120, and the called binary file calls another binary file 110 . The interdependence between binary files (and subsystems) grows very complex. *Because of the complex layers of dependence, a change 110 made in one subsystem 104 may affect other subsystems 108, 106, directly, or through a series of dependencies.* Because of this interdependence, the effect of a change may have far reaching unpredictable effects. Since the extent of dependence for any given binary file varies, the affects of all changes are not equal.

[Application, at page 5, line 26 to page 6, line 11; emphasis added.]

Applicants also note that additional language has been added to clarify the "propagating" language of the claim, in particular as it relates to interactions between multiple binary files, subsystems, and systems. At Figure 5, and its accompanying text at pages 10-13, the Application describes implementations of, for example, "recording indications of the entry and exit points in a record for the binary file," "propagating dependency information" by "generating pairs of entry and exit points for binary files which are dependent on one another," and "determining subsystem dependency information comprising entry and exit points for the one or more subsystems based on the generated pairs of entry and exit points for the set of binary files." Applicants note this in response to the Action's arguments in its Response to Arguments section.

The Echelon system described in Srivistava_2, cannot teach or suggest "generating pairs of entry and exit points for binary files which are dependent on one another" or *"marking unchanged logical abstractions which are in one or more unchanged binary files," where "the unchanged logical abstractions [are] dependent on marked changed logical abstractions in the one or more changed binary files;" because the Echelon operates only on a single binary file and does not show dependencies between files.* In contrast to the claim language and the descriptions above, the Echelon system described in Srivistava_2 does not operate on multiple binary files. Instead, the Echelon system, at most, operates only on *two versions of the same binary program* to find differences between the versions:

> Echelon takes as input two versions of a program in binary form along with the test coverage information of the old binary and produces a prioritized list of the test, as well as a list of modified and new blocks . . . .

[Srivistava_2, at § 3.1, page 3, 5th paragraph, left column to 1st paragraph, right column.] Indeed, this limitation of the described Echelon system is one motivation for the use of the techniques of the instant Application:

> Historically, techniques have been proposed for test selection and test prioritization to reduce the cost of testing. *However, these proposed techniques focus internally only on a program. For example, they consider internal parameters such as changes made to the program itself, rates of faults in the program, and block coverage of the program.*

[Application, at page 1, line 26 to page 2, line 3; emphasis added.] The application goes on at Example 10 to provide additional description of differences between the technologies:

> At 1308 , the method propagates the changes to compute the affected parts of the system by performing analysis at each of three levels of abstractions—binary, subsystem, and system. For example, as discussed in view of FIG. 14 , the propagation determines what basic blocks depend on the marked basic block. The blocks that depend directly or indirectly on a marked (affected) basic block are marked during propagation. *This information (marked blocks) is used, for example, to determine how an affected basic block might affect an unchanged basic block in another subsystem.* In one case, this information is used to exercise tests that execute unchanged basis blocks that depend on affected blocks elsewhere in the system.
>
> Prior to the described technology, unchanged basic blocks within a program did not receive consideration for risks or testing, because the

> information that the unchanged block depended on a changed block in another subsystem was unknown. This propagation of dependency information marks these unchanged blocks so they can be exercised accordingly, or so risks can be evaluated properly.

[Application, at page 20, lines 13-26; emphasis added.] Applicants, however, find no description in Srivastava_2 of reviewing dependencies about more than one binary file, as Sirvastava_2 is solely concerned with comparing versions of a single file and thus not dependencies *between files*.

Furthermore, Applicants note a description of the MaxScout system, which utilizes one implementation of the claimed technology, as superior to the Echelon system described in Srivistava_2. This description can be found in the 2005 paper "Efficient Integration Testing using Dependency Analysis," ("the 2005 publication") which is included in the presently filed IDS. Applicants respectfully note that Amitabh Srivastava and Jay Thiagarajan who are the listed inventors of the instant application *and* are the authors of the Srivastava_2 reference *are also the authors of the 2005 publication*.

The 2005 publication clearly shows that the technology of claim 21 was not found in the system described in Srivastava_2:

> MaxScout may mark a program as affected or changed even though there were no changes in any of its own code. All the previous systems including Echelon were not designed for such cases.

[2005 publication A, at § 2, last paragraph; emphasis added.] The 2005 publication goes on to describe the operation of the MaxScout system in greater detail, including how it differs from Echelon:

> MaxScout first uses MaX dependency information for each binary to mark call-in points that have a path to the affected block. *Next, MaxScout uses MaX to conduct reachability analysis to determine the call-in points in other binaries that can reach an affected call-in point. If a call-in point of a binary has been affected, MaxScout marks the binary as affected.*
>
> *MaxScout prioritizes the available tests for each affected binary. (Note that a binary may be affected even if not a single block was changed.)* MaxScout uses Echelon [20] to prioritize tests. MaxScout, however, uses a different algorithm to compute the impacted blocks. *In Echelon [20], the impacted blocks were computed by finding the set of blocks that were either modified or were*

> *added. MaxScout, on the other hand, defines the impacted blocks as a set of call-out blocks of the binary that are connected to affected call-in points.* If a call-out point is affected, all its dependent call-in points are affected. We thus prioritize tests that cover an affected call-in and call-out point over others; a test which covers more call-in and call-out points will get a higher priority. *Unlike Echelon's measure, MaxScout's measure addresses tests for binaries that have been affected even though they were not modified.*

[2005 publication A, at § 6, 2nd and 3rd paragraphs; emphasis added.] As the 2005 publication shows, the Echelon system described in Srivastava_2 does not teach or suggest "generating pairs of entry and exit points for binary files which are dependent on one another," or "marking unchanged logical abstractions which are in one or more unchanged binary files," where "the unchanged logical abstractions [are] dependent on marked changed logical abstractions in the one or more changed binary files" as recited in claim 21. Applicants further note that even if, for the sake of argument, the MaxScout system described in the 2005 publication were determined not to be covered by the language of claim 21, this would not change that the 2005 publication demonstrates that Echelon cannot teach or suggest each and every element of claim 21. Applicants therefore request that the rejection of claim 21, as well as its dependent claims 22-24, which recite separately patentable material, be withdrawn and that the claims be allowed.

Claim 25

Claim 25, as amended, recites, in part:

> means for determining binary dependencies, comprising pairs of entry and exit points, for binary files in a defined system;
>
> means for propagating the binary dependencies to identify binary files dependent on other binary files in other subsystems;
>
> . . .
>
> means for propagating marked changes using the determined and propagated dependencies to unchanged binary files; and
>
> means for prioritizing tests which cover the unchanged binary files based on test coverage of marked changes and propagated marked changes;

For at least the reasons given above with respect to claim 21, Srivastava_2 does not describe at least the above-quoted language of claim 25. Srivastava_2 thus does not teach or suggest every limitation of claim 25. Applicants therefore request that the rejection of claim 25, which recites separately patentable material, as well as its dependent claim 26, be withdrawn and that the claims be allowed.

Claim 29

Claim 29, as amended, recites, in part:

> marking logical abstractions changed from a previous version in one or more changed binary files;
>
> propagating marked changes according to the dependency information comprising marking unchanged logical abstractions in one or more unchanged binary files dependent on marked changes in the one or more changed binary files;

For at least the reasons given above with respect to claim 21, Srivastava_2 does not describe at least the above-quoted language of claim 29. Srivastava_2 thus does not teach or suggest every limitation of claim 29. Applicants therefore request that the rejection of claim 25, which recites separately patentable material, as well as its dependent claims 30 and 31, be withdrawn and that the claims be allowed.

Double Patenting Rejection

The Action maintains its provision rejection of claims 21, 25, and 29 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over the '053 application in view of Srivastava_2. Applicants respectfully submit the claims in their present form are patently distinct over the cited art.

In the course of the provisional double patenting rejection, the Action admits that '053 application claims do not recite a number of features, including:

> determining dependency information about binary files, propagating such information to determine subsystem and system dependency, marking changed and unchanged logical abstraction to prioritize tests.

[Action, at page 3.] Instead, the Action finds these features in Srivastava_2.

Applicants respectfully note that, for at least the reasons given above with respect to claims 21, 25, and 29, Srivastava_2 does not teach or suggest at least one of the features of each claim as amended. For example, Srivastava_2 does not teach or suggest "generating pairs of entry and exit points for binary files which are dependent on one another," or "marking unchanged logical abstractions which are in one or more unchanged binary files," where "the unchanged logical abstractions [are] dependent on marked changed logical abstractions in the one or more changed binary files" as recited in claim 21. Because language from each of claims 21, 25, and 29 is found neither in the '053 application claims nor in Srivastava_2, such language is not taught or suggested by the combination of the two.

For at least this reason, the Action fails to establish a *prima facie* case of obviousness over the '053 Application in view of Srivastava. Accordingly, applicants request that the provisional double patenting rejection be withdrawn.

Interview Request

If any issues remain, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.

Conclusion

The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Gregory L. Maurer/
Gregory L. Maurer
Registration No. 43,781